Regular Correspondence:
195 North Harbor Drive, Suite 5403, Chicago Illinois 60601-7540

Docketed Correspondence:
Post Office Box 7131, Chicago Illinois 60680-7131

Peter K. Trzyna, Esq.

Telephone: (312) 240-0824   Facsimile: (312) 240-0825

E-mail: pktlaw@email.msn.com

RECEIVED
CENTRAL FAX CENTER

JAN 2 6 2005

Fax

To: Examiner Patrice Winder	**Re:** 09/399,578 IDS and Cited Art
Firm: United States Patent and Trademark Office	**Date / Time:** January 26, 2005
Street Address:	**Phone:** (571) 272-3935
City, State Zip: Washington, D.C., 20231	**Fax:** (703) 872-9306
cc:	**No. of Pages:** 22 (including cover)

PRIVACY AND CONFIDENTIALITY NOTICE

The information contained in this communication is confidential and may be legally privileged. It is intended solely for the use of the individual or entity to whom it is addressed and other authorized to receive it. If you are not the intended recipient, you are hereby notified that any disclosure, copying, distribution or taking of any action in reliance on the contents of this information is strictly prohibited. If you received this communication in error, please immediately notify us by a collect telephone call to the writer at the writer's direct number indicated above, and return the original message and documents to the sender at the above address via the United States postal service.

Message:

ksfo-client.txt

```
#!/usr/local/bin/perl

# WebChat(tm) Client v 0.2
# Copyright (c) 1995 Internet Roundtable Society
# Programmed by Michael Fremont, email: webchat@irsociety.com

#      This program is free software; you can redistribute it and/or modify
#      it under the terms of the GNU General Public License as published by
#      the Free Software Foundation; either version 2 of the License, or
#      (at your option) any later version.

#      This program is distributed in the hope that it will be useful,
#      but WITHOUT ANY WARRANTY; without even the implied warranty of
#      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#      GNU General Public License for more details.

#      You should have received a copy of the GNU General Public License
#      along with this program; if not, write to the Free Software
#      Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.


# this script is executed when a user submits the "chat" form.

#LITERALS
#--------
$LOCK_SH = 1;
$LOCK_EX = 2;
$LOCK_NB = 4;
$LOCK_UN = 8;

$TRUE    = 1;
$FALSE   = 0;

#GLOBALS
#-------
$client                                 = "http:/cgi-bin/nph-client#anchor1";

# the following line is for the www.irsociety.com system
#$talkfile                              = "/home/webchat/transcripts/ksfo";
# the following line is for the webchat.service.digital system
#$talkfile                      =
"/usr/local/httpd/htdocs/webchat/transcripts/ksfo";
$talkfile                       =
"/home/lanshark/www/pages/webchat/transcripts/ksfo";

$point_gif                      =
"http://www.cybertoday.com/cybertoday/webchat/point.gif";
$webchat_logo                   =
"http://www.ccnet.com/laporte/images/ksfologo.gif";
$about_local_server                     = "http://www.cybertoday.com/";
$local_tz                               = "PDT (-0700 GMT)";

$read_block_size                        = 512;
$num_context_paras                      = 0;
$num_context_paras_when_starting        = 10;
$para_mark_size                         = 7;     # 7 digits (space padded)
$back_jump                              = 10;
$way_back_when                          = 40;

# get the form
&ReadParse;
```

Page 1

```
                          ksfo-client.txt
$last_read_para = $in{last_read_para};
$wants_dates_printed = $in{wants_dates_printed};
$back_para = $in{back_para};

# we've changed to letting you specify how far back you want to scroll
# we're leaving the old code in case we go back
if($in{Back} > 0)
        {
        $back = $TRUE;
        $back_para = $back_para - $back_jump;
        ($back_para < 1) && ($back_para = 1);
#       $last_read_para = $back_para;
        $last_read_para = $last_read_para - $in{Back};
        ($last_read_para < 1) && ($last_read_para = 1);
        }



#!!! for debugging
#if($in{InputText} eq "")
#       {
#       $in{handle} = "Michael Fremont";
#       $in{InputText} = "hello, world!";
#       }

# open a (properly initialized) transcript file

# later: if need to die, put text in some log file somewhere
open (TRANSCRIPT, "+<$talkfile") || die "Client
can't open transcript file";

# If the user input any text, add it to transcript file
($in{InputText} ne "" && $back eq "") && &add_to_transcript;

# Update the output area or send error message if nothing new
&output_new_form;
exit;


sub output_new_form

# if there is new output for this user, send it to him.  Otherwise return
# "no new info" error message to his browser so it keeps his current state

{
local ($buf, $tbuf, $last_para, $found, $amount_to_read);
local ($para_num, $date, $handle, $headURL, $headURLsize, $text);



# look for first context paragraph that is to be output to user
# note - it may not be in transcript file anymore, if the file was pruned
# but for this version, assume pruning has not yet been implemented

$first_context_para = $last_read_para - $num_context_paras;
($first_context_para < 1) && ($first_context_para = 1);
$ffirst_context_para =
        "\0" .
        pack("A$para_mark_size", $first_context_para) .
        "\1";

# read a block from the end of the file and look for the context para.
```
Page 2

```
                                  ksfo-client.txt
# If not found, read more from the file until it is found

$first_read = $TRUE;

seek(TRANSCRIPT, 0, 2); #move to EOF

# loop until found context para or at beginning of file
# (BOF determined by finding para 0 mark at beginning of buf)
while ( !($found = $buf =~ /$ffirst_context_para/))
        {
        $amount_to_read = $read_block_size;
        $curr_loc = tell(TRANSCRIPT);

        ($curr_loc == 0) && last; # read to beginning of file already

        ($curr_loc < $read_block_size) && ($amount_to_read = $curr_loc);
        seek(TRANSCRIPT, -$amount_to_read, 1);
        read(TRANSCRIPT, $tbuf, $amount_to_read);
        if($first_read == $TRUE)
                {
                # if no new data, exit loop
                # check by looking for same para# at end of file as before
                $first_read = $FALSE;
                $last_para=
                        pack("A$para_mark_size", $last_read_para+1);

                $next_para_num =
                        substr($tbuf, length($tbuf)-$para_mark_size-1);

                # commented out so we ALWAYS return data - otherwise
                # NETSCAPE can get confused in its caching. (I think)
#               ($next_para_num eq $last_para)
#                       && last;

                # if the context para is way before $next_para_num, it's
                # because the user just got into chat, and the transcript
                # file is big.  Change context para so they only get
                # stuff near the end of file, and aren't overwhelmed.
                if(($last_para lt $next_para_num - $way_back_when) && !$back)
                        {
                        $first_context_para = $next_para_num -
$num_context_paras_when_starting;
                        $ffirst_context_para =
                                "\0" .
                                pack("A$para_mark_size", $first_context_para) .
                                "\1";
                        }
                }

        # add new stuff to $buf
        $buf = $tbuf . $buf;

        # move back to before the data we just read
        seek(TRANSCRIPT, -$amount_to_read, 1);
        }

# send "no data" status if:
#       * no new data, or
#       * there's no data at all in the file
if(!$found)
        {
        #!! the next line must change based on which ver. of HTTP req. came in
        print "HTTP/1.0 204 NO RESPONSE\n";
                                Page 3
```

```
                                    ksfo-client.txt
           print "Server: WebChat Client via CERN/3.0\n";
           print "Content-Type: text/html\n\n";
           exit;
           }

print "HTTP/1.0 200 OK\n";
print "Server: WebChat Client via CERN/3.0\n";
print "Content-Type: text/html\n\n";

&output_form_header;

#print context

# break the buffer into separate paragraphs
# paragraph 0 has file-global info, and is not a real para so don't output it

# NOTE: this will could result in a lot of elements if we read a lot of
# the file
@paras = split(/\000/, $buf);

# output each paragraph
for($a=1; $a<$#paras; $a++)
           {
           &print_para($a);
           }


print "<FORM ACTION=\"$client\" METHOD=\"POST\">\n";


$last_read_para = $next_para_num -1;
&output_hidden_field("last_read_para", "$last_read_para");

($back eq "") && ($back_para = $last_read_para);
&output_hidden_field("back_para", "$back_para");

&output_form_trailer;
}


sub print_para

# prints the indicated paragraph to the user

{
($out_para) = @_;

$anchor_para = $last_read_para;
($anchor_para < 1) && ($anchor_para = 1);

($para_num, $date, $handle, $headURL, $headURLsize, $text) =
           split(/\001/, $paras[$out_para]);

           # put the anchor one paragraph in front of new stuff
           if ($para_num == $anchor_para)
                    {
                    print "<A NAME=\"anchor1\"></A>";
                    }

           if($para_num == $last_read_para+1)
                    {
                    print "<IMG ALIGN=bottom SRC=\"$point_gif\"><BR>\n";
                    }
```

Page 4

ksfo-client.txt

```
($headURL =~ /http:\/\/\w/) && print $headURL;


# the BR CLEAR=left causes our version of Air Mosaic to really puke, but
# without it we can't wrap the text to the right of the images, which really
# saves screen space and looks a lot better.  So we'll leave it in until
# we get lots of complaints from users.
#         print "$handle:", "<BR>\n$text<BR><P>\n";
          print "$handle: . . . . $date", "<BR>\n$text<BR CLEAR=left><P>\n";
}


sub output_form_header

# prints the header portion of the talk form (everything up to the chat section)

{
print    "<HTML>\n",
         "<HEAD>\n",
         "<title>webchat</title>\n",
         "</HEAD>\n",
         "<BODY>\n",
         "<IMG ALIGN=bottom SRC=\"$webchat_logo\"><HR>\n"; }


#FORM FOLLOWS (mostly)

sub output_form_trailer

#prints the trailer portion of the talk form (everything after the chat section)

{
print
         "<BR><INPUT TYPE=\"submit\" NAME=\"Chat\" VALUE=\"Chat\">  Get/Send message
\n",
         " . . Scroll Back \n<INPUT TYPE=\"text\" SIZE = \"5\" NAME=\"Back\">
messages",
         "<TEXTAREA NAME=\"InputText\" ROWS=3 COLS=70></TEXTAREA><P>\n",
         "Your Handle:   <INPUT NAME=\"handle\" VALUE=\"$in{handle}\"><p>\n",
         "Your picture URL:   <INPUT TYPE=\"text\" SIZE = \"50\" NAME=\"picture\"",
         "VALUE=\"$in{picture}\"><P>\n",
#        "Your room:   <SELECT NAME=\"room\">\n",
#        "<OPTION>Room 1\n",
#        "<OPTION>Room 2\n",
#        "<OPTION>Room 3\n",
#        "<OPTION>Room 4\n",
#        "<OPTION>Room 5\n",
#        "<OPTION>Room 6\n",
#        "</SELECT><P>\n",
         "</FORM>",
         "Click here for <A
HREF=\"http://www.irsociety.com/webchat/help.html\">Help</A>.   ",
#        "Click here for a <A
HREF=\"http://www.irsociety.com/webchat/transcript.html\">Transcript.</A>\n",
#        "Click here for <A
HREF=\"http://www.irsociety.com/webchat/options.html\">Options</A>",
         "*Go to the <A HREF=
\"http://www.irsociety.com/webchat/webchat.html\">WebChat Home Page</A>\n",
         "*About this <A HREF= \"$about_local_server\">Server</A>\n",
#        "<FORM>",
#        "<INPUT TYPE=\"submit\" NAME=\"exit\" VALUE=\"Goodbye\">",
#        "</FORM>",
```

Page 5

```
                                    ksfo-client.txt
        "</BODY>",
        "</HTML>";
}


sub init_transcript_file

# makes a new transcript file with the given name, and initializes it as
# follows:
#
# a dummy paragraph, numbered 0, that looks like this:
#       0       \1
#       possibly some global file info here, not yet defined
#       \0
#       1       \1
#
# the first real paragraph in a transcript file is numbered 1
#
# paragraphs have the following format:
# (all entries are \1 terminated, except the null paragraph terminator)
#
# paragraph number (7 digits, space padded)
# date
# handle
# head URL
# head URL size
# text
# null paragraph terminator

# paragraph numbers are fixed in size so we can improve efficiency.
# The end of a transcript file always has the next paragraph number so we
# can very quickly see if there is any new text.  By having it be a fixed
# size field we know exactly where the beginning of it is.

# Records are null terminated so we can easily split the file into
# paragraphs, and fields are terminated with \1 so we can split a record.

{

}


sub output_hidden_field

# outputs an HTML formatted hidden field to stdout
# input parameters are:
#       name, value

{
local ($name, $value);
($name, $value) = @_;

print "<INPUT TYPE=\"hidden\" NAME=\"$name\" VALUE=\"$value\">\n";
}


sub add_to_transcript

# the user has submitted text to add to the conversation
# add it, as appropriate, to the transcript

{
# now we're ready to write it, get file lock
                                    Page 6
```

```
                                        ksfo-client.txt
flock(TRANSCRIPT, $LOCK_EX);     # waits here until gets exclusive lock
# analyze input for links like http: and gopher:
&analyze_input;

# the next paragraph number is already at the end of the file; get it
# stored as a null, then $para_mark_size digits string, space padded, then "\1"
seek(TRANSCRIPT, -($para_mark_size+1), 2);
read(TRANSCRIPT, $next_para_num, $para_mark_size+1);
#re-seek to end of file for OS systems that need interspersed read/seek/write
seek(TRANSCRIPT, 0, 2);
print TRANSCRIPT join("\1", $date, $handle, $headURL, $headURLsize,
        $input_text), "\1\0", pack("A$para_mark_size", ++$next_para_num), "\1";

# done with writing, unlock file
flock(TRANSCRIPT, $LOCK_UN);
}


sub analyze_input

# looks at the text input from the user; any hyperlink references found
# (such as http: and gopher:) are converted to HTML so they become live
# when sent back to a user. NOT YET IMPLEMENTED.

# also constructs header information such as date of input, handle, etc.

{
# get the date and time
local ($sec, $min, $hour, $mday,$mon,$year, $wday, $yday, $isdat) = localtime;
local ($am_pm, $picture);

$am_pm = "AM";
if ($hour > 12)
        {
        $hour = $hour -12;
        $am_pm= "PM";
        }
($min < 10) && ($min = '0' . $min);
local (@day_of_week) = ("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat");
local (@month) = ("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep",
"Oct", "Nov", "Dec");
$date = "$day_of_week[$wday], $month[$mon] $mday, $hour:$min$am_pm $local_tz";

# get the user's handle
$handle = $in{handle};

# convert user's head URL to HTML form
$picture = $in{picture};

# if the machine has a domain name, make it inline
if ($picture =~ m!(http://[a-z][\-~a-z0-9/\.]+\.gif\w*)!i)
        {
        ($headURL = "<IMG ALIGN=left HSPACE=10 SRC=\"" . $picture . "\">\n");
        }

# otherwise, just point to it as a hotlink
elsif ($picture =~ m!(http://[\-~a-z0-9/\.]+\.gif\w*)!i)
        {
        ($headURL = "<A HREF=\"$picture\">picture</a><br>\n");
        }

#get the size of the user's picture
#not yet implemented
```

Page 7

```
                                   ksfo-client.txt
$headURLsize=1;

# look for hyperlinks in inputted text and convert them to HTML
# not yet fully implemented
$input_text = $in{InputText};

# look for HTML markup language in the input.  If they get it wrong, it will
# cause many browsers to die, so for now, we disallow it.
# Brain-dead, temporary solution:  if we find pairs of <> brackets, replace
# them with the words 'less than' and 'greater than' and put out an
# 'advisory' message.
# We do this rather than trying to parse HTML now.  Later we will.
# side-effect: if the user really was using  '<' and '>' for something else
# in the message... oops.

$input_text =~ s/<([^<]*)>/\1/gi;

#this is such a hack that if anyone claims I did it I will deny it!
# inline any .gifs referenced in user's text
# first make them invisible to the next line
$input_text =~ s!http(://[\-~a-z0-9_/\.]+\.gif\w*)!xxxx$1!gi;
$input_text =~ s!http(://[\-~a-z0-9_/\.]+\.jpg\w*)!xxxx$1!gi;

# make any other hypertext pointers live
#known bug: matches http://www.irsociety.com. (includes the period in button)
$input_text =~ s!(http://[~\-a-z0-9_/\.]+\.\S+)!<A HREF=\"$1\"><B>button</B></A>!gi;
$input_text =~ s!(ftp://[~\-a-z0-9_/\.]+\.\S+)!<A HREF=\"$1\"><B>button</B></A>!gi;
$input_text =~ s!(mailto:[@-a-z0-9_/\.]+\.\S+)!<A HREF=\"$1\"><B>button</B></A>!gi;

# now inline the .gifs and .jpgs
        # only display headURL if from a machine with a domain name, not just
        # a number.  This is so the user's browser doesn't hang up "forever"
        # trying to reach a machine that is likely not to be a permanent
        # member of the Internet

        # for pics from machines with just an IP address, use a placeholder
        # image that is live and points to the real address.  If the user
        # wants to see it, he can click on it.  If the link is dead, it
        # won't look like webchat has failed.


$input_text =~ s!xxxx(://[\-~a-z0-9_/\.]+\.gif\w*)!<IMG SRC =\"http$1\">!gi;

#$input_text =~ s!xxxx(://[\-~a-z0-9_/\.]+\.gif\w*)!<A
HREF=\"http$1\">picture</a><br>!gi;


$input_text =~ s!xxxx(://[\-~a-z0-9_/\.]+\.jpg\w*)!<IMG SRC =\"http$1\">!gi;
###
}




# Perl Routines to Manipulate CGI input
# S.E.Brenner@bioc.cam.ac.uk
# $Header: /cys/people/seb1005/http/cgi-bin/RCS/cgi-lib.pl,v 1.7 1994/11/04 00:
#17:17 seb1005 Exp $
#
# Copyright 1994 Steven E. Brenner
# Unpublished work.
# Permission granted to use and modify this library so long as the
# copyright above is maintained, modifications are documented, and
```

Page 8

```
                              ksfo-client.txt
# credit is given for any use of the library.
#
# Thanks are due to many people for reporting bugs and suggestions
# especially Meng Weng Wong, Maki Watanabe, Bo Frese Rasmussen,
# Andrew Dalke, Mark-Jason Dominus and Dave Dittrich.

# see http://www.seas.upenn.edu/~mengwong/forms/    or
#      http://www.bio.cam.ac.uk/web/                 for more information

# Minimalist http form and script (http://www.bio.cam.ac.uk/web/minimal.cgi):
# if (&MethGet) {
#    print &PrintHeader,
#        '<form method=POST><input type="submit">Data: <input name="myfield">';
# } else {
#    &ReadParse(*input);
#    print &PrintHeader, &PrintVariables(%input);
# }


# MethGet
# Return true if this cgi call was using the GET request, false otherwise
# Now that cgi scripts can be put in the normal file space, it is useful
# to combine both the form and the script in one place with GET used to
# retrieve the form, and POST used to get the result.
sub MethGet {
    return ($ENV{'REQUEST_METHOD'} eq "GET");
}

# ReadParse
# Reads in GET or POST data, converts it to unescaped text, and puts
# one key=value in each member of the list "@in"
# Also creates key/value pairs in %in, using '\0' to separate multiple
# selections

# If a variable-glob parameter (e.g., *cgi_input) is passed to ReadParse,
# information is stored there, rather than in $in, @in, and %in.
sub ReadParse {
    local (*in) = @_ if @_;

  local ($i, $loc, $key, $val);

  # Read in text
  if ($ENV{'REQUEST_METHOD'} eq "GET") {
    $in = $ENV{'QUERY_STRING'};
  } elsif ($ENV{'REQUEST_METHOD'} eq "POST") {
    read(STDIN,$in,$ENV{'CONTENT_LENGTH'});
  }

  @in = split(/&/,$in);

  foreach $i (0 .. $#in) {
    # Convert plus's to spaces
    $in[$i] =~ s/\+/ /g;

    # Split into key and value.
    ($key, $val) = split(/=/,$in[$i],2); # splits on the first =.
        #!! text fields return empty values when user doesn't use; dump these
        ($val eq "") && next;

    # Convert %XX from hex numbers to alphanumeric
                                  Page 9
```

```
                              ksfo-client.txt
    $key =~ s/%(..)/pack("c",hex($1))/ge;
    $val =~ s/%(..)/pack("c",hex($1))/ge;

    # Associate key and value
    $in{$key} .= "\0" if (defined($in{$key})); # \0 is the multiple separator
    $in{$key} .= $val;
  }

  return 1; # just for fun
}

# PrintHeader
# Returns the magic line which tells WWW that we're an HTML document

sub PrintHeader {
  return "Content-type: text/html\n\n";
}

# PrintVariables
# Nicely formats variables in an associative array passed as a parameter
# And returns the HTML string.

sub PrintVariables {
  local (%in) = @_;
  local ($old, $out, $output);
  $old = $*;   $* =1;
  $output .=   "<DL COMPACT>";
  foreach $key (sort keys(%in)) {
    foreach (split("\0", $in{$key})) {
      ($out = $_) =~ s/\n/<BR>/g;
      $output .=   "<DT><B>$key</B><DD><I>$out</I><BR>";
    }
  }
  $output .=   "</DL>";
  $* = $old;

  return $output;
}


# PrintVariablesShort
# Nicely formats variables in an associative array passed as a parameter
# Using one line per pair (unless value is multiline)
# And returns the HTML string.


sub PrintVariablesShort {
  local (%in) = @_;
  local ($old, $out, $output);
  $old = $*;   $* =1;
  foreach $key (sort keys(%in)) {
    foreach (split("\0", $in{$key})) {
    foreach (split("\0", $in{$key})) {
      ($out = $_) =~ s/\n/<BR>/g;
      $output .= "<B>$key</B> is <I>$out</I><BR>";
    }
  }
  $* = $old;

  return $output;
}

}
```

Page 10